A Duplicated Code Refactoring Advisor (DCRA)

Francesca Arcelli Fontana, Marco Zanoni, Francesco Zanoni

University of Milano-Bicocca
Department of Informatics, Systems and Communication (DISCo)
Evolution of Software Systems and Reverse Engineering (ESSeRE) Lab
Duplicated Code

Replication of source code fragments (a.k.a. «clones») within a software system

```java
public class MyClass {
    public int myMethod1() {
        int a = 0;
        int b = 1;
        a++;
        b++;
        return a + b;
    }
    public void myMethod2() {
        int a = 0;
        int b = 1;
        a++;
        b++;
        System.out.print(a + b);
    }
}
```
Consequences

Redundancy
the same feature is implemented in several parts of the system

Lower maintainability
modifications to code inside a clone may require modification of all its duplications
Duplication removal: Issues

Design-level:
- Abstraction changes
- Responsibility

Low level:
- Increased (!) code size
- Code adaptation activities

Removal is expensive

All choices delegated to developers

Developers usually reluctant to remove duplications
Duplication removal: Main questions

1. Which duplications should be removed?
   – not all duplication are worth removing

2. How should duplications be removed?
   – which refactoring(s) should be applied?

Refactoring: *improvement of a software system internal structure without changing its external behaviour* [Fowler, 1999]
Duplication removal: Available support

No automatic procedure can answer the two main questions

Approaches found in the literature to exclude unsuitable refactorings:

• **classification** of duplications in categories
• **definition of related refactorings** for each category
Duplication classification by clone positions

Fowler [Fowler, 1999]:
  – generic approach
  – few duplication positions

Golomingi [Golomingi, 2001]:
  – systematic approach
  – very detailed duplication positions

Giesecke [Giesecke, 2003]:
  – Golomingi’s approach adapted to Java language

Developers’ involvement is still heavy
DCRA approach

1. Extension of Golomini’s classification categories

2. Further criteria:
   - similarity level (%)
   - container kind (block, method)
   - coupling level (variables involved)

3. Automated evaluation of refactorings through a derived score/index, based on:
   - code length variation
   - OOP compliance/quality

more accurate refactoring sets

Goal: CONCRETE REDUCTION OF DEVELOPERS’ INVOLVEMENT
DCRA structure

User

Java byte code

Java source code

Clone detector

CLONE DETECTOR

Poor clone details

Clone detector report

CLONE DETAILER

Rich clone details

REFACTORING ADVISOR

Refactoring advice

REFACTORING ADVICE AGGREGATOR

Aggregated refactoring advice
DCRA parameters and design choices

Clone detector:
- block-level detection
- renamed duplications excluded

Refactoring advisor:
- full management of duplications related to the most recurrent categories among the most cohesive entities: 1) within the same class, 2) between sibling classes

Extensible architecture – allows addition of:
- duplication categories
- refactorings
- evaluation criteria
DCRA validation procedure

DCRA procedure applied on four systems of Qualitas Corpus collection [Qualitas Research Group, 2012]

Refactoring advisor output was manually examined to assess refactoring suggestion suitability
DCRA validation result (1)

The set of managed categories allowed to process more than 80% of all duplications

Managed duplications (suggestions):

- Accepted: 4/5
- Not accepted: 1/5
DCRA validation result (2)

Unaccepted suggestions:

32% within the same class
   — refactoring-unworthy duplications (*clones too short*)

68% between sibling classes
   — (In most cases) complex refactorings suggested
     • simpler refactorings were applied
     • or no change was performed at all
Conclusions

Duplicate Code Refactoring Advisor (DCRA) – automation to support decisions about:
– which duplications should be removed
– how duplications should be removed

Validation results:
– Evaluations involving duplications within the same class (most instances) achieve good accuracy
Future developments

Reassessment of refactoring evaluation criteria and detection settings:
- Minimum clone size: currently too low?
- Duplications between sibling classes: lower scores for “Form Template Method” and “Pull Up Method Object”

Extensions:
- duplication categories
- → associated refactorings
- criteria for refactoring evaluation, e.g., distinction of r, w, r/w variables
- use clone groups to avoid partial refactorings
Thank you!

marco.zanoni@disco.unimib.it
References


Clone types

<table>
<thead>
<tr>
<th>CLONE TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>identical code fragments, only white space differences allowed</td>
</tr>
<tr>
<td>Type 2</td>
<td>code fragments with identical structure and syntax, with identifier, literal and type renaming allowed</td>
</tr>
<tr>
<td>Type 3</td>
<td>code fragments with added, removed or modified statements</td>
</tr>
<tr>
<td>Type 4</td>
<td>code fragments implementing the same algorithm in different ways</td>
</tr>
</tbody>
</table>
Fowler’s approach

<table>
<thead>
<tr>
<th>SITUATION</th>
<th>SUGGESTED REFACTORIZINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same class</td>
<td>Extract Method</td>
</tr>
<tr>
<td>Sibling class</td>
<td>Extract Method, Pull Up Method, Form Template Method, Substitute Algorithm</td>
</tr>
<tr>
<td>Unrelated class</td>
<td>Extract Class, keep one code instance within the mostly related class, replacing all other instances with invocations</td>
</tr>
</tbody>
</table>
Golomingi’s approach

<table>
<thead>
<tr>
<th>SCENARIO</th>
<th>SUGGESTED REFACTORINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same method</td>
<td>Extract Method</td>
</tr>
<tr>
<td>Same class</td>
<td>Extract Method, Insert Method Call, Form Template Method</td>
</tr>
<tr>
<td>Sibling class</td>
<td>Pull Up Method, Extract Method, Substitute Algorithm, Form Template Method, Replace Subclass with Field, Extract Superclass</td>
</tr>
<tr>
<td>Superclass</td>
<td>Insert Super Call, Pull Up Method, Push Down Method, Form Template Method</td>
</tr>
<tr>
<td>First cousin class</td>
<td>Pull Up Method, Form Template Method, Extract Method, Extract Superclass</td>
</tr>
<tr>
<td>Ancestor class</td>
<td>Extract Method, Pull Up Method, Form Template Method</td>
</tr>
<tr>
<td>Common hierarchy class</td>
<td>Pull Up Method, Extract Method, Form Template Method, Extract Superclass</td>
</tr>
<tr>
<td>Unrelated class</td>
<td>Extract Class, keep one code instance only within the mostly related class, replacing all other instances with invocations</td>
</tr>
</tbody>
</table>
## Giesecke’s approach

<table>
<thead>
<tr>
<th>SITUATION</th>
<th>SUGGESTED REFACTORIZATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same class</td>
<td>Extract Method</td>
</tr>
<tr>
<td>Superclass</td>
<td>Extract Method</td>
</tr>
<tr>
<td>Common hierarchy</td>
<td>Extract superclass (if needed) + Pull up method</td>
</tr>
</tbody>
</table>
# DCRA locations and refactorings

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>REFACTORINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same method</td>
<td>Extract method</td>
</tr>
<tr>
<td>Same class</td>
<td>Replace method with method object</td>
</tr>
<tr>
<td>Sibling class</td>
<td>Merge method</td>
</tr>
<tr>
<td>Same external superclass</td>
<td>Pull up method</td>
</tr>
<tr>
<td>Superclass</td>
<td>Pull up method object</td>
</tr>
<tr>
<td>First cousin class</td>
<td>Form template method</td>
</tr>
<tr>
<td>Ancestor class</td>
<td>Leave unchanged</td>
</tr>
<tr>
<td>Common hierarchy class</td>
<td></td>
</tr>
<tr>
<td>Unrelated class</td>
<td></td>
</tr>
</tbody>
</table>
## Compared classifications

<table>
<thead>
<tr>
<th>CATEGORY</th>
<th>FOWLER</th>
<th>GOLOMINGI</th>
<th>GIESECKE</th>
<th>DCRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same method</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Same class</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sibling class</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Same external superclass</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Superclass</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>First cousin class</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Ancestor class</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Common hierarchy class</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Unrelated class</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Qualitas Corpus
location recurrences

- SAME METHOD: 3%
- SAME CLASS: 29%
- SIBLING CLASS: 8%
- SAME EXTERNAL SUPERCLASS: 19%
- SUPER CLASS: 3%
- ANCESTOR CLASS: 2%
- FIRST COUSIN CLASS: 2%
- COMMON HIERARCHY CLASS: 0%
- UNRELATED CLASS: 1%

28/05/2015